1. **Program to implement K means Clustering**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

mydata = pd.read\_csv("Mall\_Customers.csv")

print(mydata.head())

iv = mydata[['Annual Income (k$)','Spending Score (1-100)']]

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters = 5 , n\_init =10 , random\_state =0 )

kmeans.fit(iv)

print(kmeans.predict(iv))

iv['cluster']=kmeans.predict(iv)

## Vizualizing the Clusters

## Cluster Plot

plt.scatter(iv.loc[iv['cluster']==0,'Annual Income (k$)'],iv.loc[iv['cluster']==0,'Spending Score (1-100)'],s=100,c='red',label='Careful')

plt.scatter(iv.loc[iv['cluster']==1,'Annual Income (k$)'],iv.loc[iv['cluster']==1,'Spending Score (1-100)'],s=100,c='green',label='Standard')

plt.scatter(iv.loc[iv['cluster']==2,'Annual Income (k$)'],iv.loc[iv['cluster']==2,'Spending Score (1-100)'],s=100,c='blue',label='Target')

plt.scatter(iv.loc[iv['cluster']==3,'Annual Income (k$)'],iv.loc[iv['cluster']==3,'Spending Score (1-100)'],s=100,c='grey',label='Careless')

plt.scatter(iv.loc[iv['cluster']==4,'Annual Income (k$)'],iv.loc[iv['cluster']==4,'Spending Score (1-100)'],s=100,c='brown',label='Sensible')

#plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1], s=200,c='yellow',label='center')

plt.title("Results of K Means Clustering")

plt.xlabel("Annual Income")

plt.ylabel("Spending Score")

plt.legend()

plt.show()

##displaying cluster1

iv[(iv['cluster']==1)][['Spending Score (1-100)','cluster']]

1. **Program to Implement SVM**

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn import metrics

from matplotlib.colors import ListedColormap

# -------------------------------

# STEP 1: Import the dataset

# -------------------------------

df = pd.read\_csv('Social\_Network\_Ads.csv')

print(df.head())

print("Dataset shape:", df.shape)

# Select relevant features and target

# Age and EstimatedSalary are features

# Purchased is the target

X = df.iloc[:, [2, 3]].values # Columns: Age, EstimatedSalary

Y = df.iloc[:, 4].values # Target: Purchased

# -------------------------------

# STEP 2: Split into train & test

# -------------------------------

X\_Train, X\_Test, Y\_Train, Y\_Test = train\_test\_split(X, Y, test\_size=0.25, random\_state=0)

print("Training data:", X\_Train.shape)

print("Test data:", X\_Test.shape)

# -------------------------------

# STEP 3: Feature Scaling

# -------------------------------

sc = StandardScaler()

X\_Train = sc.fit\_transform(X\_Train)

X\_Test = sc.transform(X\_Test)

# -------------------------------

# STEP 4: Linear Kernel SVM

# -------------------------------

classifier\_linear = SVC(kernel='linear', random\_state=0)

classifier\_linear.fit(X\_Train, Y\_Train)

Y\_Pred\_linear = classifier\_linear.predict(X\_Test)

print("\nAccuracy Score (Linear Kernel):", metrics.accuracy\_score(Y\_Test, Y\_Pred\_linear))

# Plotting decision boundary for Linear Kernel

plt.figure(figsize=(6,4))

plt.scatter(X\_Test[:, 0], X\_Test[:, 1], c=Y\_Test, cmap=ListedColormap(('red','green')), edgecolors='k')

# Extract coefficients

w = classifier\_linear.coef\_[0]

a = -w[0] / w[1]

xx = np.linspace(-2.5, 2.5)

yy = a \* xx - (classifier\_linear.intercept\_[0]) / w[1]

plt.plot(xx, yy, 'b--')

plt.title('SVM Decision Boundary (Linear Kernel)')

plt.xlabel('Age (scaled)')

plt.ylabel('Estimated Salary (scaled)')

plt.show()

# -------------------------------

# STEP 5: RBF Kernel SVM

# -------------------------------

classifier\_rbf = SVC(kernel='rbf', random\_state=0)

classifier\_rbf.fit(X\_Train, Y\_Train)

Y\_Pred\_rbf = classifier\_rbf.predict(X\_Test)

print("\nAccuracy Score (RBF Kernel):", metrics.accuracy\_score(Y\_Test, Y\_Pred\_rbf))

# Visualization for RBF Kernel

plt.figure(figsize=(6,4))

X1, X2 = np.meshgrid(

np.arange(start=X\_Test[:, 0].min()-1, stop=X\_Test[:, 0].max()+1, step=0.01),

np.arange(start=X\_Test[:, 1].min()-1, stop=X\_Test[:, 1].max()+1, step=0.01)

)

plt.contourf(X1, X2, classifier\_rbf.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.scatter(X\_Test[:, 0], X\_Test[:, 1], c=Y\_Test, edgecolors='k')

plt.title('SVM Decision Boundary (RBF Kernel)')

plt.xlabel('Age (scaled)')

plt.ylabel('Estimated Salary (scaled)')

plt.show()

# -------------------------------

# STEP 6: Polynomial Kernel SVM

# -------------------------------

classifier\_poly = SVC(kernel='poly', degree=4, random\_state=0)

classifier\_poly.fit(X\_Train, Y\_Train)

Y\_Pred\_poly = classifier\_poly.predict(X\_Test)

print("\nAccuracy Score (Polynomial Kernel, degree=4):", metrics.accuracy\_score(Y\_Test, Y\_Pred\_poly))

# Visualization for Polynomial Kernel

plt.figure(figsize=(6,4))

plt.contourf(X1, X2, classifier\_poly.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.scatter(X\_Test[:, 0], X\_Test[:, 1], c=Y\_Test, edgecolors='k')

plt.title('SVM Decision Boundary (Polynomial Kernel, degree=4)')

plt.xlabel('Age (scaled)')

plt.ylabel('Estimated Salary (scaled)')

plt.show()

# -------------------------------

# STEP 7: Summary

# -------------------------------

print("\n--- Summary ---")

print("Linear Kernel Accuracy :", metrics.accuracy\_score(Y\_Test, Y\_Pred\_linear))

print("RBF Kernel Accuracy :", metrics.accuracy\_score(Y\_Test, Y\_Pred\_rbf))

print("Poly Kernel Accuracy :", metrics.accuracy\_score(Y\_Test, Y\_Pred\_poly))

1. **Python program using TensorFlow** to train a neural network on the classic **MNIST handwritten digits dataset**

import tensorflow as tf

from tensorflow.keras import layers, models

# Load dataset (MNIST digits: 28x28 grayscale images of digits 0–9)

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# Normalize pixel values (0–255) to (0–1)

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# Build a simple neural network

model = models.Sequential([

layers.Flatten(input\_shape=(28, 28)), # Flatten 28x28 images into 784 inputs

layers.Dense(128, activation='relu'), # Hidden layer with 128 neurons

layers.Dense(10, activation='softmax') # Output layer (10 classes: digits 0–9)

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))

# Evaluate on test data

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print("\nTest accuracy:", test\_acc)